МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: ИЕРАРХИЧЕСКИЕ СПИСКИ

Вариант № 16

Студент гр. 8304 Рыжиков А. В.
Преподаватель Фирсов К. В.

Санкт-Петербург
2019

1 Цель работы.

Дано логическое выражение в префиксной форме. Необходимо проверить его синтаксическую корректность.

Дополнение 2 – группа заданий 16-24.

Пусть выражение (логическое, арифметическое, алгебраическое*) представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме ((<операция> <аргументы>)), либо в постфиксной форме (<аргументы> <операция>)). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (* b (- c))) или (OR a (AND b (NOT c))).

В задании даётся один из следующих вариантов требуемого действия с выражением: *проверка синтаксической корректности*, *упрощение* (преобразование), *вычисление*.

Пример *упрощения*: (+ 0 (* 1 (+ a b))) преобразуется в (+ a b).

В задаче *вычисления* на входе дополнительно задаётся список значений переменных

$$((x_1\ c_1)\ (x_2\ c_2)\ \dots\ (x_k\ c_k)\ ),$$

где x_i – переменная, а c_i – её значение (константа).

В индивидуальном задании указывается: тип выражения (возможно дополнительно - состав операций), вариант действия и форма записи. Всего 9 заданий.

---

* - здесь примем такую терминологию: в *арифметическое* выражение входят операции +, -, *, /, а в *алгебраическое* – +, -, * и дополнительно некоторые функции.

16) логическое, проверка синтаксической корректности, добавить 4-ую операцию (которая может принимать 2 аргумента), префиксная форма

! Переменной назовём маленькую букву английского алфавита.

! Операция AND ,OR и -> определены для 1,2 и более переменных

!Операция NOT определена для одной переменной

2 Описание программы

Программа решает поставленную задачу при помощи иерархических списков и рекурсии. Рекурсивное решение

следует из того, что количество аргументов может может быть 1, 2 и более, притом что само выражение может быть аргументом. Рекурсивно заполняем элементы иерархического списка. За основу для решения задачи берём структуру Node. Пройдя иерархический список, убедимся, что выражение принадлежит к синтаксически верному, выведем положительный результат, иначе отрицательный.

3.1 Зависимости и объявление функций, структуры данных

```
#include <iostream>
#include <string>
#include <fstream>
#include <memory>

struct Node;

struct Node {
    char data;
    bool isAtom;
    std::shared_ptr<Node> next;
    std::shared_ptr<Node> list;
};

bool checkWord(std::string name);

std::string getUpdateString(std::string name, std::string removeSubstring, std::string
newSubstring);

std::shared_ptr<Node> create();

void createList(std::string name, int *count, std::shared_ptr<Node>node);

bool counting(std::shared_ptr<Node>tmp);

int getCountNode(std::shared_ptr<Node>node);

bool checkList(std::shared_ptr<Node>node);

bool isAllLower(std::string name);
```

Описание структур данных.

Структура Node содержит указатель на следующий элемент,

имеет флаг isAtom, если он положителен, то структура также хранит в себе информация в поле data, если же он отрицателен, то указатель list указывает на непустой список.

3.2 Функция main.

Программа решает поставленную задачу при помощи рекурсии и иерархических структур, чтение происходит из файла test2.txt (также возможен ввод данных вручную) .

```
int main() {

    int your_choose = 0;

    std::cout << "If you want to enter data from a file, enter \'1\'\n";
    std::cout << "If you want to enter data manually, enter \'2\'\n";

    std::cin >> your_choose;

    if (your_choose == 1) {
        std::ifstream fin;
        fin.open("C:\\Users\\Alex\\Desktop\\test2.txt");

        if (fin.is_open()) {
            std::cout << "Reading from file:" << "\n";

            int super_count = 0;

            while (!fin.eof()) {

                super_count++;

                std::string str;
                getline(fin, str);

                std::cout << "test #" << super_count << " \"" + str + "\"" << "\n";
                mainCheck(str);

            }
        } else {
            std::cout << "File not opened";
        }

        fin.close();
    } else {
        if (your_choose == 2) {
            std::cout << "Enter data \n";
            std::string str;
            std::cin >> str;
            mainCheck(str);
        }
```

```
    }

    return 0;
}
```

Функция mainCheck.

Функция проверяет являются ли поданные данные синтаксически корректными. Явным признаком успешной проверки (в случае если выражение является логическим выражением) служит то, что должно выполниться условия положительных ответов из рекурсии обхода иерархического списка .

```
void mainCheck(std::string name) {

    if (checkWord(name)) {
        name = getUpdateString(name, "AND", "1");
        name = getUpdateString(name, "OR", "2");
        name = getUpdateString(name, "->", "3");
        name = getUpdateString(name, "NOT", "4");

        if (isAllLower(name)) {


            int count = 1;

            std::shared_ptr<Node> list (create());

            createList(name, &count, list);

            std::cout << checkList(list->next);
            std::cout << '\n';
        } else {
            std::cout << "0 not ok Unnecessary capital letters in the expression";
            std::cout << '\n';
        }

    } else {
        std::cout << "0 not ok The expression is wrong";
        std::cout << '\n';
    }


}
```

3.3 Вспомогательные Функции

1)bool checkWord(std::string name);

Функция проверяет входящую строку на фатальные ошибки, наличие лишних символов, соответствие открывающих кавычек закрывающим.

```
bool checkWord(std::string name) {

    if (name.size() <= 2) {
        return false;
    }

    if (!(name[0] == '(' && name[name.size() - 1] == ')')) {
        return false;
    }

    int count = 0;

    for (char i : name) {
        if (i == '(') {
            count++;
        } else {
            if (i == ')') {
                count--;
            } else {
                if (!isalpha(i) && i != '-' && i != '>') {
                    return false;
                }
            }

        }
    }

    return count == 0;
}
```

2)std::string getUpdateString(std::string name, std::string removeSubstring, std::string newSubstring);

Функция заменяет подстроку в строке на другую подстроку.

```
std::string getUpdateString(std::string name, std::string removeSubstring, std::string
newSubstring) {

    size_t index = 0;
    while (true) {
        index = name.find(removeSubstring, index);
        if (index == std::string::npos) break;
        name.replace(index, removeSubstring.size(), newSubstring);
        index += newSubstring.size();
    }

    return name;

}
```

3)std::shared_ptr<Node> create();

Создаёт элемент Node и возвращает его.

```
std::shared_ptr<Node> create() {
    Node *newNode = new Node;
    newNode->next = (nullptr);
    newNode->list = (nullptr);
    newNode->isAtom = false;
    newNode->data = ' ';
    std::shared_ptr<Node> sharedPtr (newNode);
    return sharedPtr;
}
```

4)bool isAllLower(std::string name)

Функция проверяет, что все буквы в строке маленькие.

```
bool isAllLower(std::string name) {
    for (char i : name) {
        if (isalpha(i)) {
            if (!islower(i)) {
                return false;
            }
        }
    }
    return true;
}
```

3.4 Функция *void createList(std::string name, int *count, std::shared_ptr<Node> node)* занимается построением иерархического списка.

```
void createList(std::string name, int *count, std::shared_ptr<Node> node) {

    std::shared_ptr<Node> tmp = node;

    while (name[*count] != ')') {
        if ((isalpha(name[*count]) && islower(name[*count])) || isdigit(name[*count]))
{
            tmp->next = create();
            tmp->next->data = name[*count];
            tmp->next->isAtom = true;
        } else {
            if (name[*count] == '(') {
                tmp->next = create();
                tmp->next->list = create();
```

```
                *count = *count + 1;
                createList(name, count, tmp->next->list);

            }
        }

        *count = *count + 1;
        tmp = tmp->next;
    }
}
```

3.5 Функция *bool checkList(std::shared_ptr<Node>node)* обходит иерархический список и проверяет правильность его построения.

```
bool checkList(std::shared_ptr<Node>node) {
    std::shared_ptr<Node>tmp = node;

    if (isdigit(tmp->data)) {
        if (!counting(tmp)) {
            return false;
        };
    } else {
        return false;
    }

    tmp = tmp->next;

    while (tmp != nullptr) {
        if (!tmp->isAtom) {
            if (!checkList(tmp->list->next)) {
                return false;
            };
        } else {
            if (isdigit(tmp->data)) {
                return false;
            }
        }
        tmp = tmp->next;
    }
    return true;
}
```

3.6 Функция *bool counting (std::shared_ptr<Node>tmp)* проверяет количество необходимых аргументов для каждой операции (для AND OR 1,2 и более , для импликации только 2 аргумента, для NOT только 1 аргумент)

```
bool counting(std::shared_ptr<Node>tmp) {
    if (isdigit(tmp->data)) {
        switch (tmp->data) {
            case '1':
                if (getCountNode(tmp) >= 1) {
                    return true;
                }
                break;
            case '2':
                if (getCountNode(tmp) >= 1) {
                    return true;
                }
                break;
            case '3':
                if (getCountNode(tmp) == 2) {
                    return true;
                }
                break;
            case '4':
                if (getCountNode(tmp) == 1) {
                    return true;
                }
                break;
        }
    }
    return false;
}
```

4 Тесты

1.1 Покажем что, программа корректно работает на простых примерах :

Ответ ok

1) (ORq)

2) (ANDq)

3) (->q(NOTa))

Ответ not ok

1) (OR)

2) (->)

3) AAAA

4) (ANq)

1.2 Покажем что, программа корректно работает на простых рекурсивных примерах (операция, операнды, операнд):

Ответ ok

1) (->q(NOTa))

2) (AND(NOTc)a)

3) (ANDa(NOTb)c)

Ответ not ok

1) (ANq)

2) (ANq.

3) ANq)

4) (->24)

1.3 Покажем что, программа корректно работает на сложных рекурсивных примерах (операция, операнды, операнд):

Ответ ok

1) (OR(ORwrere(ORsdvdvd(NOT(ANDdfgh)))))

2) (->(NOTq)(OR(ORwrere(ORsdvdvd(NOT(ANDdfgh))))))

3) (ORasd(ORasdf(ORsdfg(ORdsfsdvdf))))

4) (AND(NOTq)w(ANDc)(AND(NOTq)w(ANDc)))

Вывод: *были построены иерархические списки для проверки синтаксической корректности выражения. Изучена работа рекурсии для задания иерархических списков. Были написаны тесты и проверена работоспособность программы.*

Приложение

Код программы lab2.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <memory>

struct Node;

struct Node {
    char data;
    bool isAtom;
    std::shared_ptr<Node> next;
    std::shared_ptr<Node> list;
};

bool checkWord(std::string name);

std::string getUpdateString(std::string name, std::string removeSubstring, std::string
newSubstring);

std::shared_ptr<Node> create();

void createList(std::string name, int *count, std::shared_ptr<Node>node);
```

```
bool counting(std::shared_ptr<Node>tmp);

int getCountNode(std::shared_ptr<Node>node);

bool checkList(std::shared_ptr<Node>node);

bool isAllLower(std::string name);


bool checkWord(std::string name) {

    if (name.size() <= 2) {
        return false;
    }

    if (!(name[0] == '(' && name[name.size() - 1] == ')')) {
        return false;
    }

    int count = 0;

    for (char i : name) {
        if (i == '(') {
            count++;
        } else {
            if (i == ')') {
                count--;
            } else {
                if (!isalpha(i) && i != '-' && i != '>') {
                    return false;
                }
            }

        }
    }

    return count == 0;
}

std::string getUpdateString(std::string name, std::string removeSubstring, std::string
newSubstring) {

    size_t index = 0;
    while (true) {
        index = name.find(removeSubstring, index);
        if (index == std::string::npos) break;
        name.replace(index, removeSubstring.size(), newSubstring);
        index += newSubstring.size();
    }

    return name;

}

void createList(std::string name, int *count, std::shared_ptr<Node> node) {

    std::shared_ptr<Node> tmp = node;

    while (name[*count] != ')') {
        if ((isalpha(name[*count]) && islower(name[*count])) || isdigit(name[*count]))
```

```
{
            tmp->next = create();
            tmp->next->data = name[*count];
            tmp->next->isAtom = true;
        } else {
            if (name[*count] == '(') {
                tmp->next = create();
                tmp->next->list = create();

                *count = *count + 1;
                createList(name, count, tmp->next->list);

            }
        }

        *count = *count + 1;
        tmp = tmp->next;
    }
}

std::shared_ptr<Node> create() {
    Node *newNode = new Node;
    newNode->next = (nullptr);
    newNode->list = (nullptr);
    newNode->isAtom = false;
    newNode->data = ' ';
    std::shared_ptr<Node> sharedPtr (newNode);
    return sharedPtr;
}

void printList(std::shared_ptr<Node>node) {
    std::shared_ptr<Node>tmp = node;
    while (tmp != nullptr) {
        if (tmp->isAtom) {
            std::cout << tmp->data;
        } else {
            std::cout << '(';
            printList(tmp->list->next);
            std::cout << ')';
        }
        tmp = tmp->next;
    }
}

int getCountNode(std::shared_ptr<Node>node) {
    std::shared_ptr<Node>tmp = node;
    int count = 0;
    while (tmp != nullptr) {
        count++;
        tmp = tmp->next;
    }

    return count - 1;
}

bool counting(std::shared_ptr<Node>tmp) {
    if (isdigit(tmp->data)) {
        switch (tmp->data) {
            case '1':
                if (getCountNode(tmp) >= 1) {
                    return true;
```

```
                }
                break;
            case '2':
                if (getCountNode(tmp) >= 1) {
                    return true;
                }
                break;
            case '3':
                if (getCountNode(tmp) == 2) {
                    return true;
                }
                break;
            case '4':
                if (getCountNode(tmp) == 1) {
                    return true;
                }
                break;
        }
    }
    return false;
}

bool checkList(std::shared_ptr<Node>node) {
    std::shared_ptr<Node>tmp = node;

    if (isdigit(tmp->data)) {
        if (!counting(tmp)) {
            return false;
        };
    } else {
        return false;
    }

    tmp = tmp->next;

    while (tmp != nullptr) {
        if (!tmp->isAtom) {
            if (!checkList(tmp->list->next)) {
                return false;
            };
        } else {
            if (isdigit(tmp->data)) {
                return false;
            }
        }
        tmp = tmp->next;
    }
    return true;
}

bool isAllLower(std::string name) {
    for (char i : name) {
        if (isalpha(i)) {
            if (!islower(i)) {
                return false;
            }
        }
    }
    return true;
}
```

```
void mainCheck(std::string name) {

    if (checkWord(name)) {
        name = getUpdateString(name, "AND", "1");
        name = getUpdateString(name, "OR", "2");
        name = getUpdateString(name, "->", "3");
        name = getUpdateString(name, "NOT", "4");

        if (isAllLower(name)) {


            int count = 1;

            std::shared_ptr<Node> list (create());

            createList(name, &count, list);

            std::cout << checkList(list->next);
            std::cout << '\n';
        } else {
            std::cout << "0 not ok Unnecessary capital letters in the expression";
            std::cout << '\n';
        }

    } else {
        std::cout << "0 not ok The expression is wrong";
        std::cout << '\n';
    }


}


int main() {

    int your_choose = 0;

    std::cout << "If you want to enter data from a file, enter \'1\'\n";
    std::cout << "If you want to enter data manually, enter \'2\'\n";

    std::cin >> your_choose;

    if (your_choose == 1) {
        std::ifstream fin;
        fin.open("test2.txt");

        if (fin.is_open()) {
            std::cout << "Reading from file:" << "\n";

            int super_count = 0;

            while (!fin.eof()) {

                super_count++;

                std::string str;
                getline(fin, str);

                std::cout << "test #" << super_count << " \"" + str + "\"" << "\n";
                mainCheck(str);
```

```
            }
        } else {
            std::cout << "File not opened";
        }

        fin.close();
    } else {
        if (your_choose == 2) {
            std::cout << "Enter data \n";
            std::string str;
            std::cin >> str;
            mainCheck(str);
        }
    }

    return 0;
}
```